

1

Selection Sort

EASY

What You Will Learn

- ✓ How Selection Sort scans for the smallest element each round
- ✓ Why it always does the same number of comparisons regardless of input
- ✓ How to write it cleanly in Java
- ✓ Where it performs well — and where it doesn't

Intuition First

Think of it like picking players for a team. You scan the whole line, find the shortest person, and put them first. Then scan the rest, find the next shortest, put them second. Repeat until done.

The Core Idea

- In each pass, find the MINIMUM element from the unsorted portion
- Swap it to the front of the unsorted section
- The sorted section grows by 1 after every pass

Visual Explanation

Start: unsorted array

Initial array — all unsorted

64	25	12	22	11
----	----	----	----	----

Pass 1 — Find minimum (11) at index 4, swap with index 0:

Scanning — 64 vs 11

64	25	12	22	11
----	----	----	----	----

↕ swap 64 ↔ 11

After Pass 1 — 11 is in place ✓

11	25	12	22	64
----	----	----	----	----

Pass 2 — Find min in [25, 12, 22, 64] → 12, swap with index 1:

Finding next min...

11	25	12	22	64
----	----	----	----	----

After Pass 2 — [11, 12] sorted ✓

11	12	25	22	64
----	----	----	----	----

■ ■ How It Works — Step by Step

1. Start with i = 0 (first position of unsorted section)
2. Assume arr[i] is the minimum. Set min_idx = i
3. Scan from i+1 to end. If arr[j] < arr[min_idx], update min_idx = j
4. After scan, swap arr[i] with arr[min_idx]
5. Move i forward by 1. Repeat until i reaches second-to-last position

■ Dry Run on [64, 25, 12, 22, 11]

Step 1

Pass 1: min=11 found at idx 4 → swap with idx 0

64	25	12	22	11
----	----	----	----	----

Step 2

Pass 2: min=12 found at idx 2 → swap with idx 1

11	25	12	22	64
----	----	----	----	----

Step 3

Pass 3: min=22 found at idx 3 → swap with idx 2

11	12	25	22	64
----	----	----	----	----

Step 4

Pass 4: min=25 stays at idx 3 (no swap needed)

11	12	22	25	64
----	----	----	----	----

Step 5

Done! Array fully sorted

11	12	22	25	64
----	----	----	----	----

■ Java Implementation

```
public static void selectionSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        // Assume current position has minimum
        int minIdx = i;
        // Scan rest of array for actual minimum
    }
}
```

```
for (int j = i + 1; j < n; j++) {
    if (arr[j] < arr[minIdx]) {
        minIdx = j; // Found smaller, update
    }
}

// Swap only if minimum is not already in place
if (minIdx != i) {
    int temp = arr[minIdx];
    arr[minIdx] = arr[i];
    arr[i] = temp;
}
}
```

■ Time & Space Complexity

Case	Time	Space	What happens?
Best	$O(n^2)$	$O(1)$	Even if sorted, still scans everything
Average	$O(n^2)$	$O(1)$	Always $n*(n-1)/2$ comparisons
Worst	$O(n^2)$	$O(1)$	Reversed array — same as best!

■ Think Like This

- If $n = 10 \rightarrow \sim 45$ comparisons
- If $n = 100 \rightarrow \sim 4950$ comparisons
- Double the input, 4x the work. That's $O(n^2)$ in action.

■ Common Mistakes

- Forgetting that the outer loop goes to $n-1$ (not n)
- Not tracking `min_idx` properly — don't reset it each inner iteration
- Swapping even when `min_idx == i` (unnecessary, wastes time)

■ Pro Tips

- Selection sort does at most $n-1$ swaps — great when writes are expensive
- Not stable — equal elements might get reordered

- Never use for large inputs; $O(n^2)$ kills performance at $n > 10,000$

■ Practice Problems

[Easy] Sort [29, 10, 14, 37, 13] using selection sort manually

[Easy] Count total comparisons for an array of size 6

[Med] Modify to sort in descending order

[Med] Find the kth smallest element using selection sort logic

■ Quick Revision

- Find minimum from unsorted section, swap to front
- $O(n^2)$ always — no best case optimization
- $O(1)$ space — in-place sort
- At most $n-1$ swaps — good for write-limited systems
- Not stable